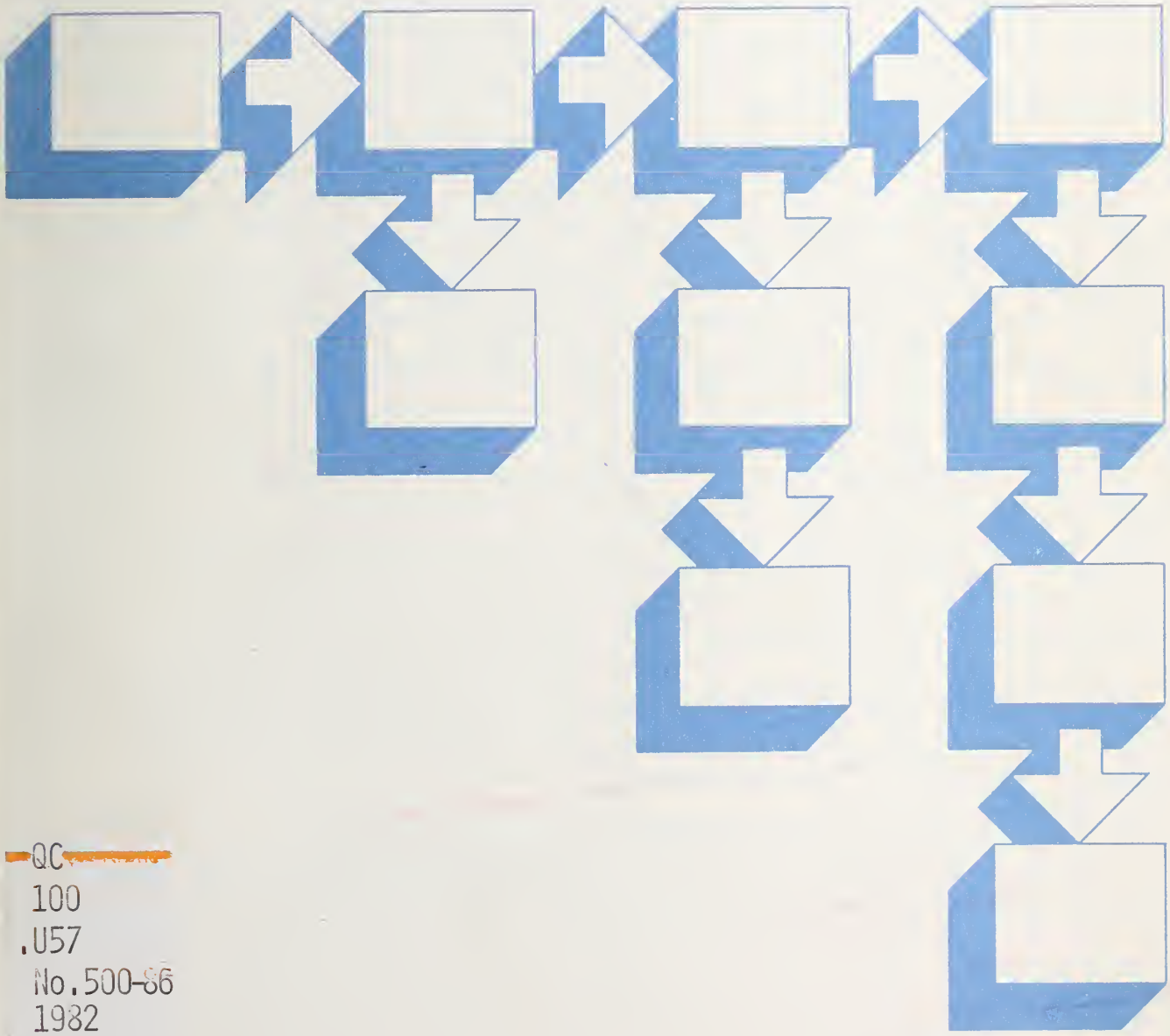
U.S. Department
of Commerce

National Bureau
of Standards

Computer Science and Technology

NBS Special Publication 500-86

An Architecture for Database Management Standards



NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, and the Institute for Computer Sciences and Technology.

THE NATIONAL MEASUREMENT LABORATORY provides the national system of physical and chemical and materials measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; conducts materials research leading to improved methods of measurement, standards, and data on the properties of materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

Absolute Physical Quantities[2] — Radiation Research — Thermodynamics and Molecular Science — Analytical Chemistry — Materials Science.

THE NATIONAL ENGINEERING LABORATORY provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

Applied Mathematics — Electronics and Electrical Engineering[2] — Mechanical Engineering and Process Technology[2] — Building Technology — Fire Research — Consumer Product Technology — Field Methods.

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following centers:

Programming Science and Technology — Computer Systems Engineering.

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Washington, DC 20234.
[2]Some divisions within the center are located at Boulder, CO 80303.

Computer Science and Technology

NBS Special Publication 500-86

An Architecture for Database Management Standards

Prepared by the

Computer Corporation of America
575 Technology Square
Cambridge, MA 02139

for the

Center for Programming Science and Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, DC 20234

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

National Bureau of Standards
Ernest Ambler, Director

Issued January 1982

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

Library of Congress Catalog Card Number: 81-600174

National Bureau of Standards Special Publication 500-86
Nat. Bur. Stand. (U.S.), Spec. Publ. 500-86, 52 pages (Jan. 1982)
CODEN: XNBSAV

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON 1982

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402
Price $3.25
(Add 25 percent for other than U.S. mailing)

- PREFACE -

In partial fulfillment of its obligations under the Brooks Act (PL 89-306), the Institute for Computer Sciences and Technology (ICST) of the National Bureau of Standards (NBS) is developing Federal Information Processing Standards (FIPS) for Database Management Systems (DBMS's). The Brooks Act and the executive orders that implement it empower ICST to help Federal agencies improve the cost-effectiveness of selecting, acquiring, and using computer resources. The applications of Federal agencies vary so widely that a single kind of DBMS would not be suitable for all of the Government. ICST's Center for Programming Science and Technology is therefore developing a family of database standards to minimize the Government's data processing costs by reducing data conversion problems, permitting more efficient use of files, increasing the portability of programs, reducing the need to retrain staff, and reducing the time and cost of developing new software and maintaining existing programs.

Standardizing DBMS's requires an architecture to ensure integration of family members, to help ICST decide how to use scarce resources most effectively for the Government as a whole, and to guide ICST's participation in voluntary standardization activities. In 1979 ICST contracted with the Computer Corporation of America to develop such a framework. This report consists of a proposed architecture for DBMS standards. As directed by ICST, the contractor has developed a functional decomposition of the DBMS standards family. Like many previous efforts in the area of database standards, this work has focused on identifying and specifying interfaces.

This document is a contractor's report. The architecture described herein will provide one input to future FIPS for database management. Because the architecture is still evolving and other inputs will be considered, the opinions and ideas expressed here do not necessarily have final NBS/ICST endorsement.

ICST solicits comments on the contractor's proposed architecture. Government agencies, commercial vendors, independent consultants, and other interested parties wishing to respond to issues raised by the report should contact:

Chief
Data Management and Programming Languages Division
Tech A-255
National Bureau of Standards
Washington, DC 20234

Table of Contents

Table of Figures

AN ARCHITECTURE FOR DATABASE MANAGEMENT STANDARDS

Computer Corporation of America

This report describes the current status of an Institute for Computer Sciences and Technology project on architectures for Database Management Systems. An architectural framework for developing DBMS standards is presented. It addresses requirements of both the Federal data processing community and the DBMS vendor community. The architecture groups DBMS functions into both internal and external components and proposes for these components a family structure that supports the integration of DBMS standards for multiple data models.

Key words: Database; database function; database management system; data model; schema; standards; system architecture; system components.

1. INTRODUCTION

1.1 Purpose of the Report

This report describes the current status of an Institute for Computer Sciences and Technology project on architectures for Database Management Systems. A proposed architectural framework, developed by the Computer Corporation of America (CCA), is presented for review by Federal Government data processing personnel, computer hardware and software vendors, and other interested groups and individuals.

This "strawman" architecture is intended for use in developing Federal Information Processing Standards for Database Management Systems (DBMS's). The architecture is referred to as a strawman in that the review of this report and other feedback will be used to modify and refine the proposed architecture.

In this section we review the requirements for DBMS standards and the status of the strawman architecture. Section 2 presents an overview of the architecture. Section 3 discusses the salient details of the architecture. Finally, Section 4 presents the implication of the architecture on the standardization of Database Management Systems.

1.2 The Need for DBMS Standards

Federal Information Processing Standards (FIPS) for Database Management Systems (DBMS's) would provide important benefits. In particular, the standards would benefit both the Federal Automatic Data Processing (ADP) departments and the computer industry. For example, the selection and procurement of computer hardware and software systems by Government data processing departments often involves extensive evaluations of dissimilar DBMS's. While standards exist for character sets, programming languages, and certain hardware interfaces, no such standards exist for DBMS's. Yet DBMS's, with a wide variety of features and performance, are one of the most critical types of Federal system procurements. The establishment of standards for DBMS's would greatly facilitate these procurements.

Standards for DBMS's would encourage computer manufacturers and commercial system developers to provide compatible, but competitive DBMS's. Currently, vendors must design and provide "complete" Database Management Systems with their own array of dissimilar functions and interfaces. DBMS standards would be used by vendors to direct the basic design of their systems; the vendors could then concentrate on performance improvements, innovative interfaces, and price reductions.

While standards for DBMS's are needed, they are difficult to develop due to the variation in existing Database Management Systems. Modern DBMS's are complex hardware and software systems that are used for a wide variety of applications. These varied and complex systems have different features and interfaces. DBMS's differ in

the levels of their interfaces: they can provide interfaces to programming languages or interfaces to languages for non-computer specialists. DBMS's also differ in the power of the data management functions they provide; some provide operations for retrieving or modifying individual records, while others provide operations for retrieving or modifying selected groups of records. DBMS's differ in a variety of other characteristics including: the physical access paths that are supported; the provisions for allowing simultaneous updating of the database by multiple users; the variety of programming languages such as FORTRAN and COBOL that may use a DBMS; and the different data models such as relational, hierarchic, and network that can be used to represent an agency's information.

A generic DBMS architecture is needed to provide in-depth unification to the wide variation of functions and components of Database Management Systems. The complexity of DBMS's can be represented by an architecture that classifies functions into components, and identifies both external and internal interfaces to these components. For example, file access functions could be grouped into a standard internal component, while output specification and formatting could be grouped into a standard external report writer component. The strawman architecture is proposed to meet this need.

A generic DBMS architecture must also accommodate multiple data models. Functions and languages common to relational, hierarchic, and network data models must be identified for each component type. For example, some query language functions are available in DBMS's for each of the three models mentioned above. Other functions and languages exploit the complexity or simplicity of a given model. A family of components in the strawman architecture is proposed for encapsulating the similarities among the data models and for supporting the required model dependent extensions.

The architecture must also allow for the systematic evolution of standards for Database Management Systems. The architecture should accommodate the prioritization of standardization efforts based on the needs of the Federal ADP community and the state-of-the-art in database technology. Future standard developments can proceed as Federal usage priorities and technology stabilizations permit.

1.3 Requirements for Database Standards

Standards for DBMS's must satisfy a diverse range of requirements for both the Federal ADP community and the vendors that supply Database Management Systems. The requirements for each of these groups are discussed below.

1.3.1 The Federal ADP Community

The standards must facilitate the selection, procurement, and use of a Database Management System for a given application environment. These requirements impacted the design of the strawman architecture. They include:

1) Facilitate the Selection of the Best DBMS.
   An architectural framework for DBMS standards should facilitate the comparisons of systems that emphasize different features for different applications. The standards must support a variety of systems to meet the respective needs of different applications.

2) Facilitate the Procurement of DBMS's.
   Standards for Database Management Systems are required so that procurement requests can directly reference the standards instead of being forced to specify their own DBMS requirements. The architectural impact is that the standard should guide the alternative decisions to be made by the procurement departments. In particular, the standards must be complete enough that ancillary specifications are generally not required.

3) Preserve Investment in Future Applications and User Training.
   The training and application development investment required to use a DBMS continues to increase. DBMS standards must therefore support commonality and evolution between different applications in order to share procedures (and policies) in the database programs and to reduce the learning time required for a new application or a new Database Management System. An implication of this objective is that the standard must be sufficiently complete so that most application programs can be written entirely with constructs that are part of the standard.

   Note that this objective is directed towards future application and training since no one type of existing DBMS dominates either the Government ADP departments or commercial users. Current DBMS

installation statistics indicate that over 20 different systems are widely in use on large mainframe computers [CW 1981].

1.3.2 DBMS Vendors

For the DBMS industry, standards should encourage competition, facilitate the introduction of DBMS products, and allow for the systematic evolution of DBMS features. These requirements also impact the design of an architecture and are reviewed below.

1) Encourage Competition and Innovation.
Standard DBMS interfaces for programming languages and interactive query languages would provide a well-defined target (and market) for DBMS vendors. Implementors could concentrate on cost/performance tradeoffs instead of proliferating end user interface designs.

In addition, if internal interfaces were standardized, creative special purpose interfaces could be built on top of the standard internal interfaces. The standardization would insure that such interfaces would run on a number of hardware and software systems. In this way, the existence of a standard supports rather than suppresses innovation.

The strawman architecture can be seen to motivate three different types of DBMS vendors. Some vendors would choose to supply standard internal interfaces through combinations of hardware and software components. Other vendors could supply the standard external interfaces to COBOL and FORTRAN programs, report writers, and query languages. Still other vendors could supply non-standard special purpose external interfaces that conform on their use of an internal interface.

2) Facilitate the Evolution of DBMS Standards.
Rather than freezing DBMS's, the architectural framework must be prepared for upward compatible changes. The architecture should identify which interfaces are currently suitable for standardization and which should not yet be standardized. The architecture should have well-defined avenues for extension of the defined facilities of the system. The extensions can be implemented first as part of specific systems. If proven beneficial, the extensions can be standardized for a given data model. If feasible (and desirable)

the extensions can be standardized for all data models. Extensions can be fed through the standardization process and cause the standards themselves to evolve in controlled and systematic directions.

3) Facilitate the Introduction of Standards.
The Federal Government is dependent upon the vendor community to produce systems which comply with Federal Information Processing Standards. To be effective, it is important that procuring agencies have an adequate selection of systems that comply with the standard. The successful introduction of the standard is, therefore, dependent upon the willingness of a sufficient number of vendors to produce compliant systems. Emphasis on this requirement encourages the adoption of a standard that supports the functionality, language styles, and data models prominent in existing systems since that will reduce the vendor investment required. Emphasis on this requirement also encourages the concept of a family of standards so that systems designed for different types of applications can still be considered standard.

1.4 Status of the Strawman Architecture

The strawman architecture is based on several key features including the comprehensive identification of DBMS functions, the grouping of those functions into components, the support of multiple data model standards, and the specification of both internal and external interfaces. Each of these key features, independently of one another, will contribute to the development of effective DBMS standards.

In developing the architecture, CCA first collected an extensive list of functions that are found in existing and proposed DBMS's. Next, criteria based on efficiency and data model independence considerations were established for analyzing the functions. These criteria were then used to collapse and group the functions into components that could be effectively packaged as products, possibly by independent producers.

More detailed specification of this DBMS architecture is in progress; this report does not describe a final product. CCA is currently in the process of refining the components and providing detailed descriptions of their functions and interfaces. The main purpose of this report

is to allow for a review of the strawman architecture's key features by the computing industry and the Federal ADP community.

2. A COMPONENT AND DATA MODEL ARCHITECTURE

The strawman architecture groups DBMS functions into both internal and external components and defines for these components a hierarchical family that supports the integration of standards for multiple data models. The identification of components in the architecture is discussed in Section 2.1. The major groupings of those components and their respective interfaces are described in Section 2.2. The key internal interface to the core database handler with the standardized internal access to the schema of schemas is discussed in Section 2.3. Section 2.4 describes how the hierarchical family of components supports the multiple data models.

2.1 DBMS Components and Interfaces

The building blocks of the strawman architecture are components and their interfaces. The components represent groupings of related DBMS functions and are designed to be potentially separate DBMS products. Each component is characterized by a collection of interfaces through which 'users' can activate the functions of the component.

The 'users' of a component can be both end users or other components of the architecture. The end user interfaces, the conventional targets of standardization activities, include direct interfaces to users and application programs. Standardization of these interfaces would permit users and application programs to be transferred from product to product without expensive training and conversion costs.

When an interface to a component is used only by another component of the architecture, it is called an internal interface. Standardization of these internal interfaces would permit and encourage the growth of a plug compatible DBMS industry. Database Management Systems involve a large number of external interfaces. Requests made through these interfaces are, in the course of their execution, eventually translated to associated retrieval

and update operations on some physical storage medium. By defining and standardizing key internal interfaces, different vendors could develop and market different or competing external interfaces. Similarly, vendors could separately market internal components that employed innovative hardware and software technologies.

Figure 2.1 shows an example fragment of a component architecture. The two circles represent components. The square inserts, labelled with numbers on the perimeter of each circle, represent interfaces for each component to its users. Those interfaces are the mechanisms through which the intended functions of the component are exercised. A line coming into one of these interfaces represents a use of that interface either by an end user, labelled "E", or by another component.

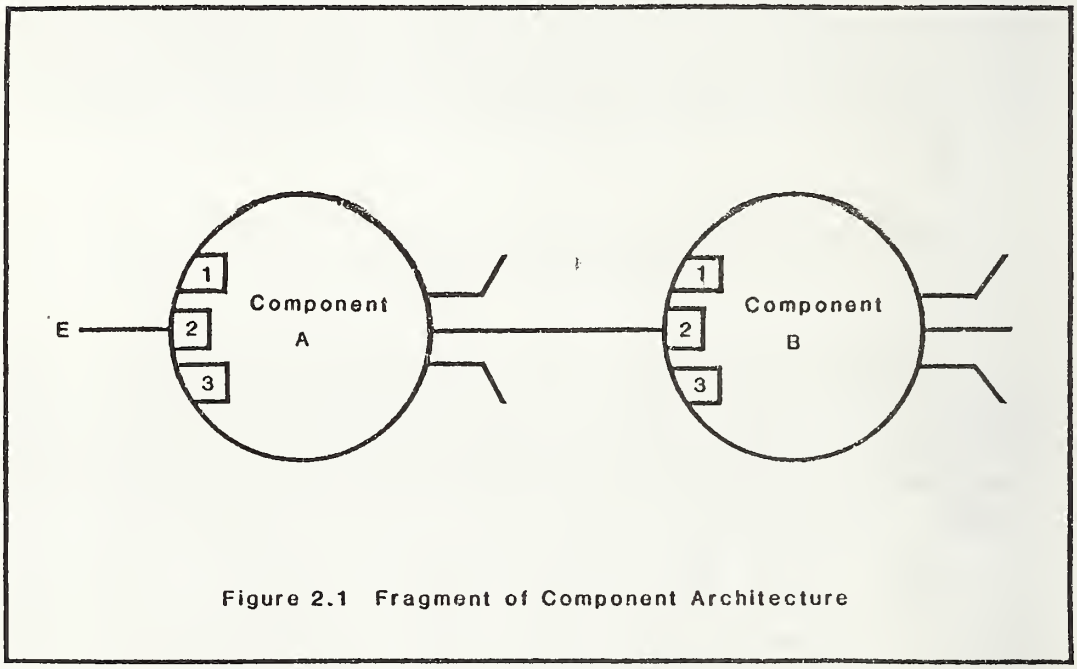


Figure 2.1 Fragment of Component Architecture

In this example there are two components. Component A provides services to an end user and could be used for schema maintenance, database administrator aids, database retrievals, etc. Component B, on the other hand, performs some functions that are needed by Component A in order to satisfy a user request. Those internal functions could be for file access, sequencing through database objects or higher level set oriented functions. The important concept is that Component A must invoke those functions through the defined interface to Component B.

When a connection is used in the architecture to show the use of one component interface by another component, this interconnection is a necessary part of the architecture. That is, the architecture requires that the function of the one component be accomplished by employing the services of another component.

The component architecture provides a basis for partitioning the development and use of DBMS standards. Some components and interfaces are presently suitable candidates for standardization. Other components represent features that are just beginning to appear in DBMS's and may be candidates for standardization at a later date. The selection and procurement of a DBMS could be based on only a subset of the standardized components. Similarly, a vendor could market a selected subset of components and, in fact, could choose to market only external interfaces.

In summary, the strawman architecture identifies each of the components of a Database Management System in terms of three factors: (a) the functions that those components perform, (b) the interfaces that each component has to its users, and (c) the necessary connections of each component to other components. In the next section an overview and classification of these components is presented.

2.2 The Major Components of the Architecture

The strawman architecture consists of four major component groups. This partition, shown in Figure 2.2, consists of schema processors, core database handler, user interfaces, and database administrator aids.

An overview of each of the four groups is presented below. In Section 3, more details on the individual components are presented.

A. The Schema Processors. Schema processors are a collection of components for handling all aspects of database definition, including structure definition, substructure definition, access control, integrity control, storage structure definition, device media control, logical performance measurement specifications and physical performance measurement specifications. The schema processors provide language processing capabilities to translate user requests, validate them, and then initiate activation of all user operations via the core database handler.

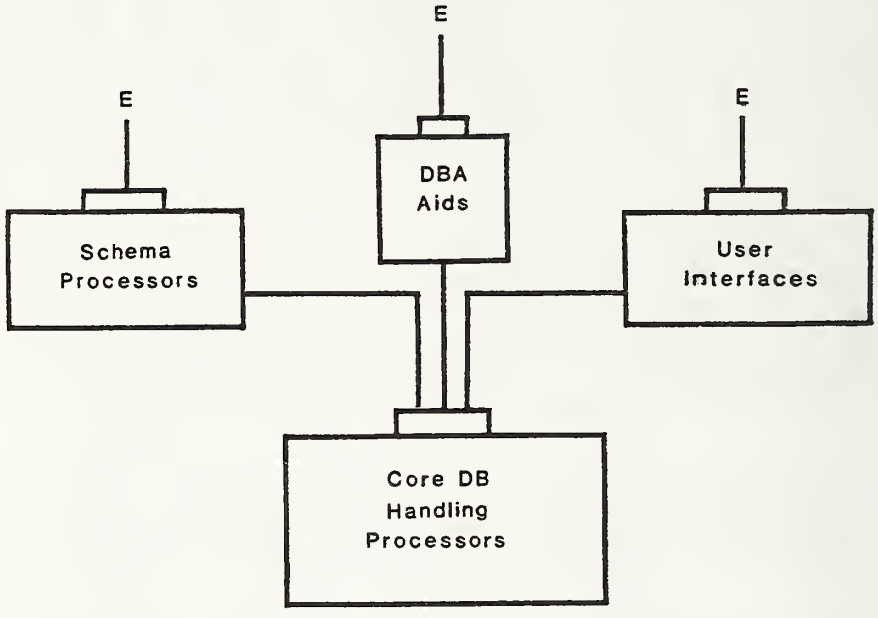


Figure 2.2 Major Conceptual Component Classes of the Architecture

B. User Interfaces. This portion of the architecture includes the components for query language processing, host language interfacing and other more advanced user interfaces, such as natural language interfaces and tools such as editors to assist in the construction of user requests.

C. Database Administration Aids. This portion of the architecture includes the data dictionary and database design aids. Both of these functions require information that is defined by the schema processors and stored by the core database handler.

D. The core database handler. These components are the central elements of the component architecture. They provide the storage and retrieval facilities for all data stored in the system: that is, both user data and system data (such as schemas) that are required by other components of the system. The interface to the core database handler is a logical interface that provides functions for manipulating both sets of records and individual records. In addition to providing (and defining) a target interface for query language processors, the core database handler also provides facilities to support interfaces to host language application programs and to schema processors. Since the latter typically deal with the database a "record-at-a-time", the requirements for the packaging of data being retrieved, stored, or updated are quite different from those of typical query processors.

The first three types of components communicate with the database through the core database handler and communicate with each other indirectly by reading and updating internal schema data through the core database handler. The rationale for this integral part of the architecture is discussed in the next section.

2.3 The Core Database Handler

The core database handler and its management of the schema of schemas are integral parts of the architecture as shown in Figure 2.3. The database handler performs all of the storage, retrieval, and updating of the user data; maintains the internal schema data; and interfaces to the operating system. The schema processors, DBA aids, and user interfaces, send commands and receive data inside a package called an envelope.

The presence of a well-defined interface to the core database handler is an unusual and significant feature of the architecture. The more typical DBMS architecture will include specific interfaces between host languages, query

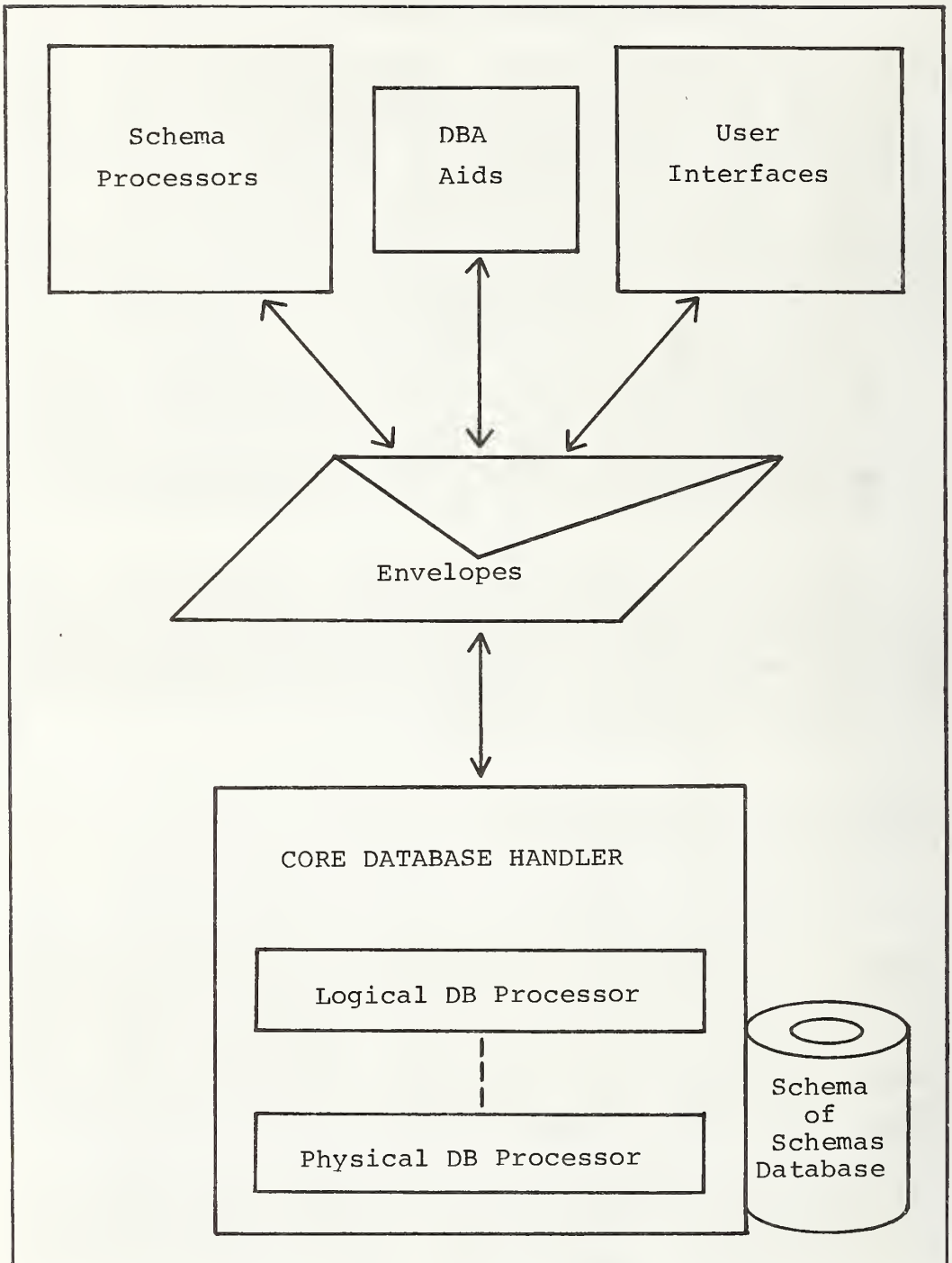


Figure 2.3 Core Database Handler

language processors, report writers, etc., and even the more "internal" parts of the system. These various end user interfaces are not translated into a single well-defined, common interface. Since there is no defined target for the user interface below the level of already existing user interfaces, it is practically impossible for any organization other than the vendor of the DBMS to create new user interfaces. With the proposed architecture, new user interfaces can be created that employ the same core database handling facilities that are used by already existing user interfaces.

The use of an envelope to communicate with the core database handler serves several purposes. The external interfaces can package multiple commands together with error condition exits and alternatives in an envelope and give the envelope to the core database handler. The core database handler can use the set of commands for multiple command concurrency control, optimization, and data streaming and return a set of data to the requesting process in an envelope. Note that if the set of data required by a given component cannot be directly specified using the core database handler functions, a covering envelope can be specified to request a larger set of the organized data. The final data selected can then be controlled by the requesting component. For example, a user may require records with a given fourth character in a certain field. If the core database handler did not directly support that functionality, it could retrieve the set of records having that character anywhere in that field. The external interface component could check the retrieval records and only return to the user the required records.

Users of the core database handler depend upon standard access to the data definitions input through the schema processors; the database administration aids and the user interfaces reference the schema data through the core database handling facilities. The external components thus need to know the logical structures in which the schema information can be obtained. If we allowed these logical structures to be invented by the creators of the schema processors, the schema information would be, in effect, inaccessible to the creators of other components. An inaccessible or non-standard internal schema representation would not permit compatible components from different vendors. Hence the component architecture requires a standard access to the schema data. The strawman architecture proposes that the schema information be stored in a schema of schemas database in order to standardize access to that information and still provide extremely flexible access to it. The combination of this architectural feature and the well-defined core database handler

interface makes it possible to have separate components for handling each different type of schema and multiple components for handling database administration aids and user interfaces.

The need for a standardized internal logical interface and standardized functions for retrieving information about the schema can be illustrated by a simple example. Suppose a DDL and COBOL DML were standardized. The standard had facilities for defining a record and items in that record. The DML had language constructs for sequencing through the records. Now suppose a user, another vendor, or a standards committee wanted to build a general purpose query language (or report writer) on top of this standard. The query language processor would allow a user to specify record names and item names that are to be printed. The implementor of that interface would be faced with two problems. First, at runtime, the query language processor would need to check the existence of an item name in a record and determine the type of the item. The type of the item is needed since character strings are printed differently than integers or packed decimal numbers. Without an internal schema of schemas, this information cannot be determined at runtime. Without a standardized internal schema of schemas, the query language would be implementation specific.

The standardization of the DML only in terms of programming language constructs causes a similar problem. It is unreasonable to expect the query language processor to generate standard DML and invoke the compiler (and possibly the precompiler) in response to a runtime request. Again, a standardized internal interface for the logical DML functions at the core database handler level would be needed to develop non-vendor specific interfaces.

The standard interface at the core database handler level to the user data and schema data is needed to provide the flexibility and extensibility to DBMS systems and standards. This interface illustrates the power of standardizing internal interfaces. Additional flexibility and extensibility through the strawman architecture are provided through component families that are discussed in the next section.

2.4 Data Models and Component Families

To be effective in directing the development of DBMS standards, the architecture must support multiple data models. The data model of a Database Management System consists of the data structures, the operations on those structures, and the update-controlling integrity constraints. Different database systems support different data models for reasons of implementation history and because different data models are best suited to specific applications. In standardizing DBMS's, this valuable diversity should be preserved.

This approach gives rise to the concept of a *family*: a related set of standard DBMS's, the members of which may differ but which are similar whenever possible.

This concept is thus intended to ensure that different members of the family of standards are similar wherever possible. This objective is achieved through the notion of the *component type*: a set of components, all members of which perform the same basic task. Members of a component type generally differ in data model. The two functional elements shown in Figure 2.1 actually denote component types rather than components. The concept of the component type makes the assumption that the same basic kinds of functions need to be performed in a Database Management System independent of data model. So the same component types appear in DBMS's that support different data models.

The concept of a component type allows us to create a hierarchy of standards, as illustrated in Figure 2.4. At the first level of the hierarchy, Level A, the family of standards is represented by a set of component types. At the next level down, Level B, we find individual instances of each component type. For example, if a component type is a query language processor, then at Level B we may find component standards for a CODASYL query language processor, a relational query language processor, and so on. At Level C, we find individual components; that is, implementations of component standards.

The concept of a family of standards is created by having different aspects of interface specifications standardized at each level. At Level A, elements of syntax and semantics that should not vary across data models are established. For example, the syntax for arithmetic or the syntax for comparison operators (less than, greater than, etc.) may be standardized. At Level B, language elements that vary from data model to data model are

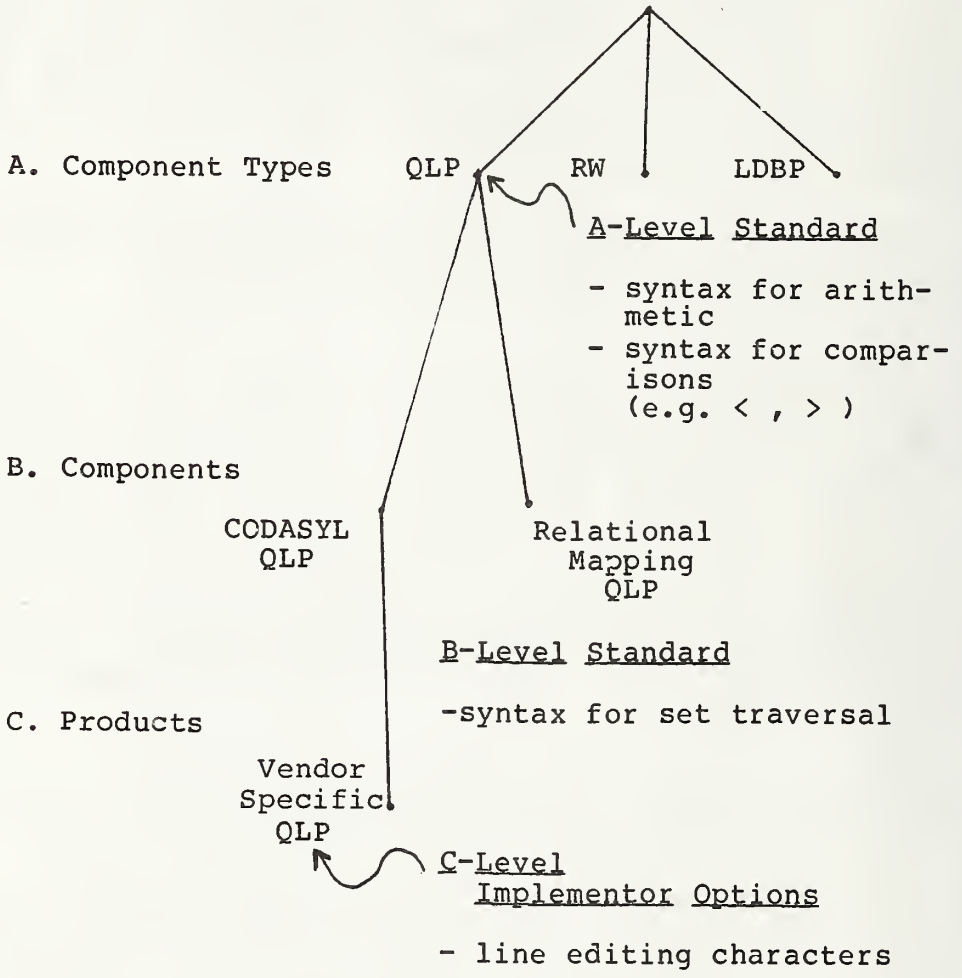


Figure 2.4 Family of DBMS Standards

specified. For example, the syntax for set traversal would be established in a network-set oriented query language but not in a relational query language. Finally, at Level C, we find the options that are left to the end implementors (e.g., line editing characters for an interactive query language processor). This notion of a family based on component types then ensures that, to the maximum extent possible, members of the family are similar; that is, wherever they can be the same, they are. Members differ only with regard to the facilities that are particular to their data models.

The Level A/Level B approach can be implemented by identifying common interface features of a component type and creating functions and statement fragments that would be common across all components of the type. As in Figure 2.4, the comparison operators ">" and "<" could be fixed and incorporated into all query languages regardless of data model. However, this does not guarantee that the final Level A syntax will consist of a set of statements. It could end up consisting of delimiter, arithmetic and comparison symbols, augmented by unrelated clauses. If the language being specified were a query language this would mean that it could be impossible to express a complete query using only Level A language constructs.

However, an initial examination suggests that sufficient commonality can be imposed on the individual components of a type to guarantee that a coherent Level A language could be produced. The advantages to making this additional effort are threefold:

- the possibility of having portable programs for those willing and able to restrict themselves to Level A features,
- easier program convertibility even for those who cannot, and
- a natural mechanism for language enhancement and evolution.

The third advantage is very important. Component types such as query language and schema description processors continue to evolve. Each new DBMS conference introduces new query operators and structuring primitives. Very frequently these are additions to functionality and not just changes to style. When such an innovation proves useful and achieves some amount of acceptance, attempts are made to add it to other languages of the same type.

The existence of a common core at Level A for any component type provides a natural route for assimilating such innovation. The new feature could be introduced as a level C extension and if successful (accepted by other

implementors) would be "promoted" to Level B. If useful at Level B and compatible with other data models, the functionality could be promoted to Level A, and DBMS's for all models would support this feature. This strategy can be characterized as a "star-fish" approach to accommodating technical innovation. The core of the starfish is all the Level A features. Each arm is a Level B variant that implements different features and approaches.

This approach is quite different from the "onion-skin" strategy suggested by [Date 1980] where three different DBMS approaches (the relational, hierarchic, and network) are constrained to be subsets of one another. As can be seen in Figure 2.5, if a hierarchic component contains a feature that a network component in the "onion-skin" approach does not, it must perforce be added to the network language or eliminated from the hierarchic language.

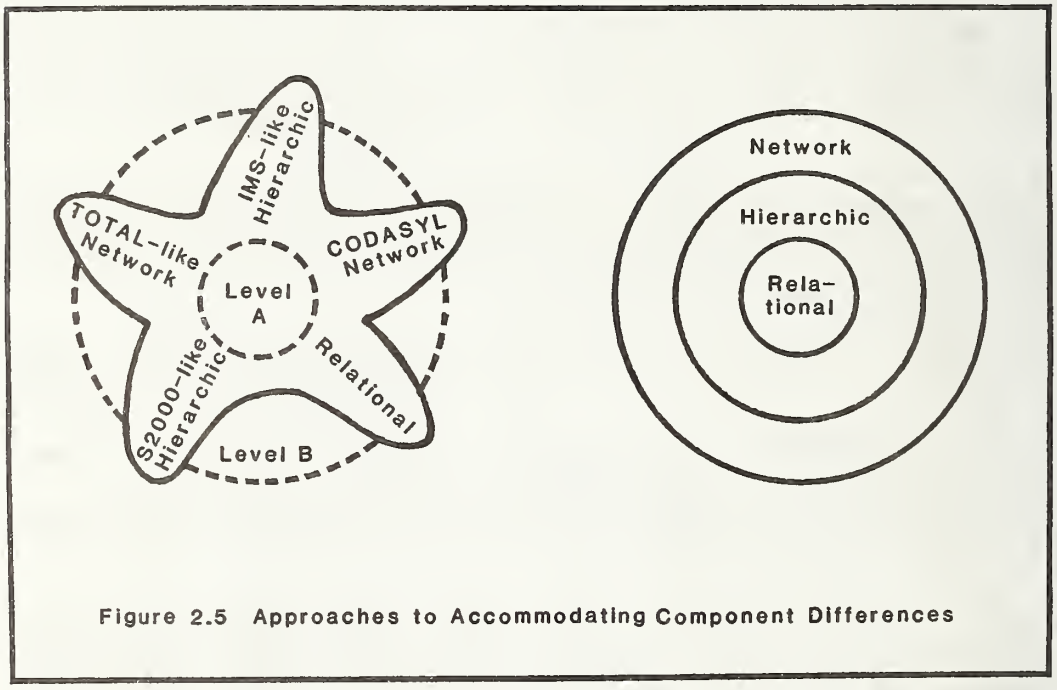


Figure 2.5 Approaches to Accommodating Component Differences

This is because the network language is simply an extension of the hierarchic language and "automatically" inherits all its features. To add a new feature using this strategy means that it must be added at the outer ring first and then pushed inward. Thus, a new feature cannot be added to just a relational component without also adding it to all the other components. Because it enforces no such inclusion discipline, the Level A/Level B approach can permit each component to have Level B

features with no counterpart in other components of that type. In this way Level B becomes the place to standardize model dependent features.

In summary, the family and component concepts both contribute to standards that can be used to integrate different approaches to database management. Furthermore, both concepts also support the evolution of such standards.

3. ARCHITECTURE DETAILS

The individual components of the DBMS architecture were designed to support a wide range of implementation strategies. Whenever the separation of a particular function into more than one component would reduce performance the separation was eliminated. The components can be placed into the four classes described in Section 2: schema processors, core database handling processors, user interfaces, and DBA aids. Within the framework of these classes, each of the individual components is briefly described.

3.1 The Schema Components

Eight schema components handle all aspects of database definition. Their positions in the architecture are illustrated in Figure 3.1. A brief summary is given of the kinds of information specified in the interface language for each of these processors.

Structure Definition

A Structure Definition Language (SDL) is used to define the structure of an application's information that is to be represented in a DBMS. The structure is defined in terms of the atomic elements and interrelationships among the atomic elements.

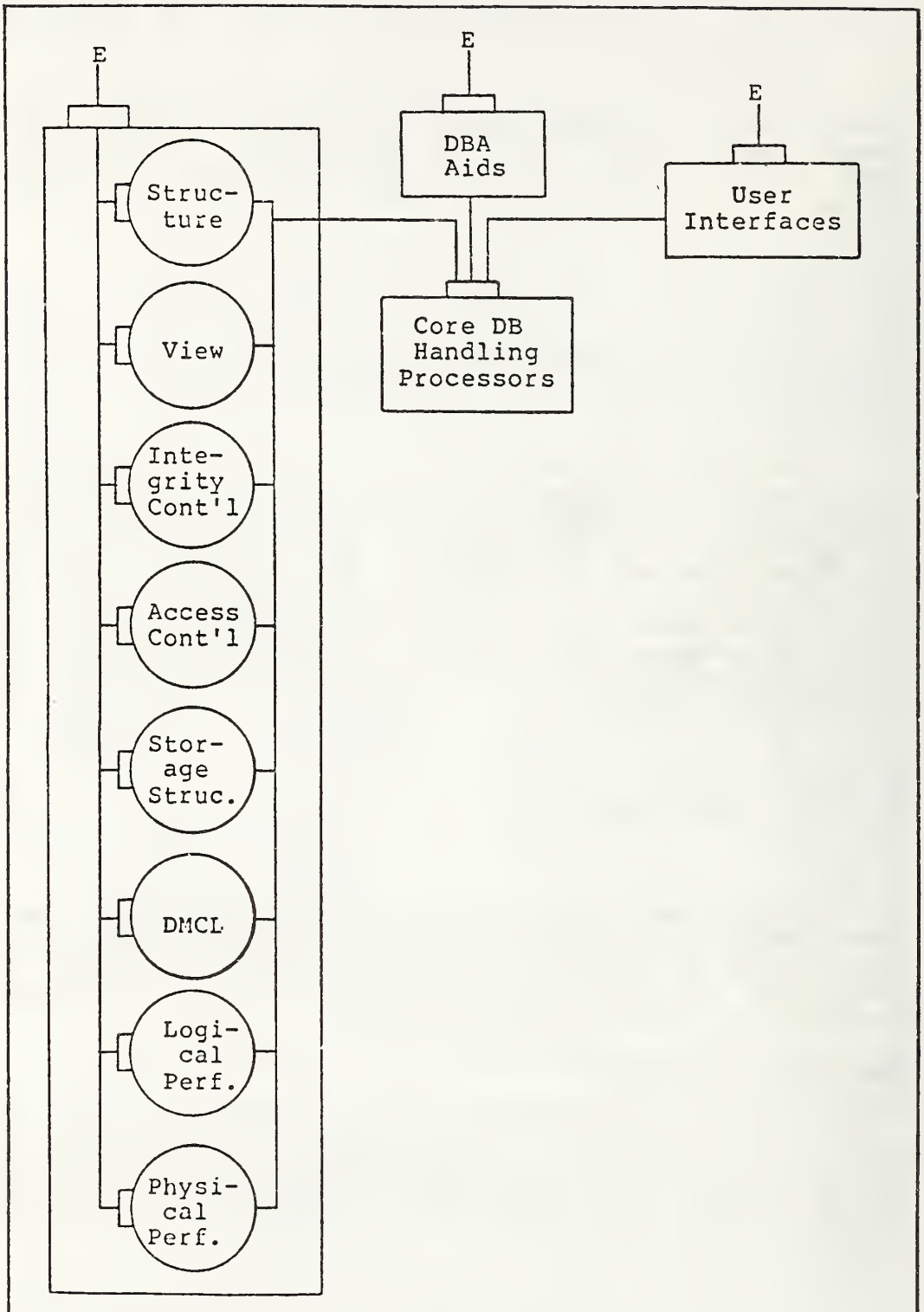


Figure 3.1 The Schema Processors

Substructure (View) Definition

A view in the DBMS Component Architecture is a manifestation of a database whose derivation can be specified by means of query language (QL) or other operations applied to structure and substructure definitions.

Integrity Control Definition

An Integrity Control Language (ICL) is used to specify constraints on a database and actions that can be taken to maintain its integrity.

Access Control Definition

An Access Control Language (ACL) is used to define for each user of the DBMS which objects he can access in what manner.

Storage Structure Definition

A Storage Structure Definition Language is used to describe for each object defined in the SDL how it is physically implemented. For example, it is used to describe how objects are positioned, the amount of space they will need, and whether they will be linked together by pointers.

Device Mapping Definition

A Device Mapping Control Language (DMCL) is used to specify how a storage structure is mapped onto physical storage media such as disks.

Logical Performance Statistics

A Performance Statistics Definition Language (PSDL) is used to specify what statistics are to be collected. At the logical level of a DBMS these can include statistics about the logical objects stored by the DBMS as well as the transactions that access these objects.

Physical Performance Statistics

The physical PSDL is used to specify what statistics are to be collected about storage and device level objects and operations.

The separation of these definition functions into individual components was motivated by the expectation that for many applications only a subset of these functions would be needed. Of the eight interface languages described above, preliminary specifications have been

developed for three. These designs are highlighted below.

3.1.1 The Structure Definition Language

A structure definition language (SDL) is used to define the structure of the application being represented in a DBMS. It defines it in terms of the atomic elements and interrelationships that the DBMS supports. The "strawman" Level A SDL proposes constructs:

a) to name atomic elements, and
b) to name and describe two levels of interrelationship: the entity and the database.

Atomic elements are called items. Items are the basic units of reference in a database. They are intuitively equivalent to CODASYL data items and relational attributes. An item is specified in the SDL by a valid name and type.

The first type of interrelationship that can be defined is the entity. An entity represents interrelationships among items. It is similar to a relational tuple and to a CODASYL record. An entity does not allow for optional items, groups, or repeating items. An entity is specified in the SDL by giving a valid name and listing the names of the items being interrelated.

The second type of Level A interrelationship is the database. The database definition collects together all entity definitions. A database definition is equivalent to a set of relation definitions and to a CODASYL schema containing only "flat" records. It is specified in the SDL by giving a valid name and listing the entity names to be included.

A "strawman" CODASYL Level B SDL would provide constructs:

a) to name and describe CODASYL style sets,
b) to name and describe CODASYL style groups, and
c) to include CODASYL style sets in the specification of a database.

A CODASYL set is called a relationship in the "strawman" Level B SDL. The features of a relationship that are specified using the SDL are the singular set option and the ordering option. At this Level B, a database structure can consist of any entities (records) and relationships (CODASYL sets) selected by the creator of the database.

3.1.2 The Integrity Control Language

An Integrity Control Language (ICL) is used to name and specify:

a) constraints on database and DBMS objects,
b) the circumstances under which a constraint is to be checked, and
c) when appropriate, the actions to be taken when a constraint is violated.

The ICL is also used to display these specifications and to delete them.

The differences between specifying structure and specifying integrity control can be subtle. In a sense, a structure definition constrains data instances to the types that have been specified in the SDL. Type (structure) declarations frequently have associated with them constraints on the effects of operations that can be applied to them. For example, relational insertions and value modifications are not permitted to introduce duplicate tuples into relations. CODASYL modifications are not permitted to change the membership of a FIXED set. Yet these same implicit constraints can be explicitly expressed in a general purpose constraint specification language such as an applied predicate logic or an appropriate query language. For the "strawman" specifications, an attempt is being made to treat all constraints uniformly by providing appropriate statements for specifying assertions.

Integrity control applies not only to user created data objects in a database but also to schema objects. For example, there may be a requirement to provide access privileges (as specified in the ACL) to some particular object only if there exists no access grant to some other object or set of objects. In this case a constraint is needed that applies to data in the access control schema rather than in some user database.

A constraint specification language is data model dependent in that constraints must be definable over structural features at both Level A and Level B. For example in the CODASYL model, a user may wish to impose the constraint that each CODASYL set of a particular type must have exactly two members or that a CODASYL group may repeat exactly three times. Since the constraint sublanguage is, in essence, a version of the query language, the same Level A/Level B decomposition exists.

Constraints can be checked in a variety of circumstances. They can be checked:

a) immediately after a primitive data manipulation operation such as an entity insertion or deletion,
b) after some logical group of changes (e.g., at the end of a transaction), or
c) upon user demand.

Only in the first case where a model dependent operation must be referenced is the specification data model dependent.

Integrity control can be either active or passive. Passive control takes the form of checking a constraint in the specified circumstances and rejecting the initiating action when the constraint is violated. Active control can take corrective actions, issue warning or reports, or notify some appropriate authority.

3.1.3 The Access Control Language

An Access Control Language (ACL) is used to specify for each object what operations can be performed by which users. This specification process is known as authorization. Once an authorization has been made and given to the core database handling processors, it is the responsibility of these processors to enforce it.

Access rights are specified in terms of:

a) the operations that can be used,
b) the data objects that can be accessed,
c) the users to whom the rights are granted,
d) any time constraints on when the rights can be used,
e) the equipment that can be used in exercising the rights,
f) legal job-mixers in which the rights can be exercised,
g) special procedures for authenticating the access, and
h) whether or not the rights can be passed on to other users.

Only the specification of the operations and the data objects are data model dependent.

Access rights to schema objects are specified with respect to the schema of schemas that is a part of the interface to the core database handling processors. These are standard formats for each different kind of information maintained by the DBMS. The schema will be designed using only Level A SDL constructs. Thus, the same ACL (and ICL) constructs can be used for this data as are used for user databases.

3.2 The Core Database Handling Processors

The core database handling processors are the heart of the architecture. They perform all of the storage, retrieval, and updating of user data, and they maintain the schema of schemas. There are three distinct, major components: the logical database processor, the physical database processor and the file access processor. Each of these components interacts closely with an associated set of components that can be classified as DBMS utilities. These utilities are provided to facilitate DBMS operation. The positions of all of these components in the architecture are illustrated in Figure 3.2.

There are three significant features embodied in this part of the architecture:

- there is a single, well-defined interface to all of the core database handling processors,
- a schema for schemas is defined and accessible through the common interface.
- there are three levels of components and their associated internal interfaces within this class.

The key concept underlying the common interface to the core database handler is the "envelope". This is described in Section 3.2.1. The schema for schemas is discussed in Section 3.2.2.

The effect of defining interfaces below the level of logical database processing provides a target for the designers of operating systems and of special purpose hardware. In this way, the strawman architecture facilitates the clean and efficient interfacing of Database Management Systems with their host machines. Without this kind of a well-defined physical level interface, it is difficult to create Database Management Systems that are both efficient and smoothly transportable.

The descriptions of these components are presented in three subsections, corresponding to each of the three major components (i.e., Sections 3.2.3, 3.2.4, and 3.2.5).

3.2.1 The Envelope

The schema processors, the database administration aids, and the user interfaces including query language processors, report writers, programming language interfaces, etc. will all use an "envelope" to request functions and receive data. An envelope to the logical data base processor will contain one or more commands to perform database functions packaged (in an envelope) as one unit. An envelope from the LDBP will contain the results of this package, including both status messages and data items.

The envelope will thus involve at least one communication from an external interface component to the LDBP and, after processing, one communication from the LDBP back to this component. These communications can be of arbitrary length. The envelope concept is similar to stored procedures that are available in several DBMS's.

An envelope must be bracketed by statements that identify the beginning and end of the sequence of commands. The brackets also associate the envelope with a user program and various status indicators and output options. An envelope with brackets is also the unit of communication for an executing user program that accesses a database. Envelopes would be generated by a user interface processor (i.e., compiler or preprocessor) in its processing of a user program.

A different number of envelopes could be generated for the user program depending on the power of the LDBP. At one extreme one envelope could be generated for each database access from the program. Alternately, multiple database accesses could be bracketed into one envelope. With a sufficiently powerful LDBP, the database accesses could be intermixed with program control logic (looping and branching) and some computations. For example, updates which take existing field values and multiply them by a constant could be accomplished entirely within the LDBP if arithmetic capabilities were present. Similarly, simple conditional power in the LDBP could reduce communications with the user program components. In the cases where complex program control logic exceeds the capabilities of the LDBP, the processing of an envelope could return more data than is actually needed.

Several alternatives exist for sending envelopes. One alternative would be to have the LDBP (and perhaps the PDBP) invoked as a separate, shared process. A second alternative would be to have the LDBP (and possibly even the PDBP) completely compiled into the host workspace.

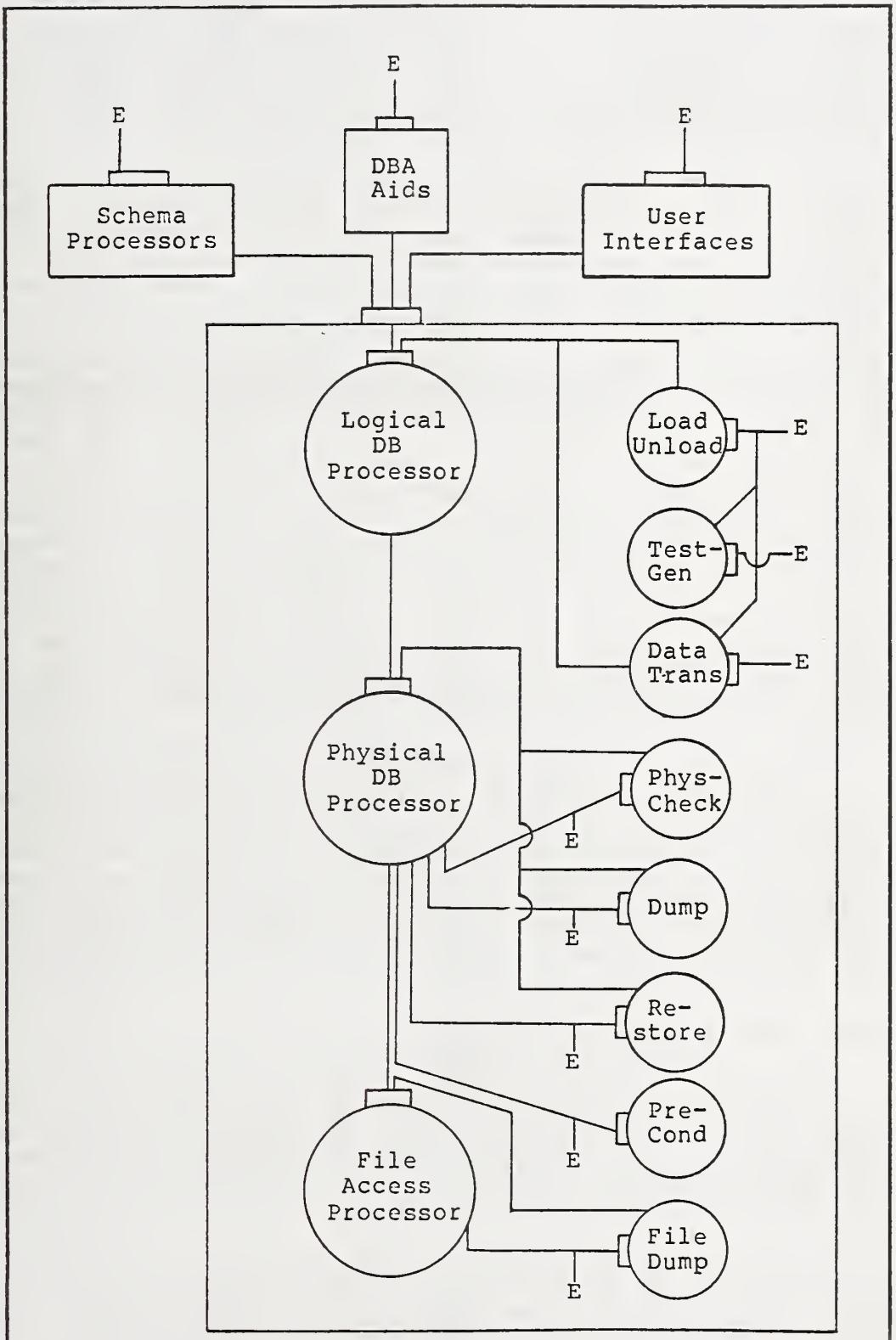


Figure 3.2 The Core Database Handling Processors

The first alternative is needed to allow the incorporation of concurrency control and recovery in the LDBP. Furthermore, unless the host process can also be trusted, the incorporation of view mapping, access control, integrity control and auditing in the host workspace is impossible because of security considerations.

In regard to the expense of message passing, it is desirable to move out of the host process as soon in the architecture as possible. Note that each communication from the host to the LDBP will, in most cases, result in one or more communications from the LDBP to the physical database processor. Similarly, each communication to the PDBP will result in one or more communications from the PDBP to the file access system. Therefore, a LDBP compiled into the host workspace could result in an increase in process message traffic between the LDBP and the PDBP or the file access system. In order to reduce the task switching required for this communication, the interprocess communication should support the general notion of pipes or variable length messages. The host system should be able to send an envelope of commands with one message to the LDBP. Similarly, the LDBP should be able to send a stream of data items and status messages back to the host with one message. In practice, on many operating systems, this type of communication is basically handled as a buffering problem: the host says send me, at most, n units worth of data; send me, at most, the next n units, etc.

However, the second alternative of having the needed LDBP and PDBP functions linked directly into the host workspace is the most efficient with regard to message passing. In this case, communication with the LDBP would be through procedure calls. Protection issues and concurrency control might be handled by trusting host processors, shared, reentrant run time libraries, or calls to the operating system.

3.2.2 The Schema for Schemas

This section describes briefly and informally the types of information that must be included in a schema database for a DBMS based on the strawman architecture. It discusses decisions that influence both the design and specification of the schema database.

The primary issue in designing the schema for schemas is that of the data model to be used. This schema is to be accessible through the same interface as user data. It would therefore be convenient if the schema were

represented using the same data model as user data. The logical structure of user data is represented in a two level data model based on the Level A/Level B concept described in Section 2. The Level A data model consists of structuring capabilities and operations that can act as a common basis for supporting any number of data model extensions. Anticipated extensions (i.e., the Level B data models) would include relational, network, and hierarchic models.

The Level A capabilities are intended to be independent of any particular Level B data model. This means that any Level B query language could be used to query a schema whose design is restricted to Level A features (since these features are a subset of the Level B features). This suggests that the design for the schema of schemas, if possible, be restricted to the Level A features.

One other schema of schemas design decision was that data dictionaries and user interfaces should be able to query the schema of schemas on any 'name' without knowing the role (entity, attribute, view, etc.) of the name within the database.

The schema for schemas contains all of the information input to the DBMS through the schema processor components. This information includes definitions for data structures, views, integrity control, access control, storage structures, device media utilization and performance monitoring as described in Section 3.1.

In summary, the schema of schemas is proposed as just another database for querying purposes. It is also proposed in the strawman architecture that the user database update operations, restricted by integrity and protection constraints, be used to update the schema databases. These operations will also trigger whatever operations are needed to create or redefine a database schema.

3.2.3 The Logical Database Handling Processors

This section describes the functionality of the logical database handling processors. These consist of the logical database processor itself, the logical loader, the logical dumper, the data translator, and the test data generator.

The Logical Database Processor

The logical database processor performs nine functions:

- logical access,
- view mapping,
- integrity control.
- access control,
- logical performance monitoring,
- concurrency control.
- restructuring,
- auditing, and
- optimization.

Each of the above functions will be described with respect to its impact on the design of the interface to this component.

Logical access is the mapping of logical data manipulation operations (DM functions) on a database to physical operations on the underlying storage structure representing that database. The DM functions are used for both user data and schema data. Syntax and protocols must be provided at the interface to express the DM functions and to name the database operands. Data retrieved by the core database handling processors must be returned through the interface to the user or requesting program.

View mapping is the transformation of logical operations on a view to logical operations on the database over which the view is defined. The same syntax can be used to specify operations on a view as is used to specify operations directly on parts of the database defined in the SDL.

Integrity control verifies that an update to a database will leave it in a consistent state with respect to some set of integrity constraints defined over the database. Constraint checking is invoked either implicitly by the operators or operands of the access request or explicitly by user invocation. Implementing implicit integrity control requires that the operators and operands of the update be known. This information is derived from the update request. Syntax is required in the LDBP interface to support explicit invocation. When the update would violate the constraints, either the access should be rejected or corrective actions (such as insertion of default values) should be taken. If the access is rejected, the LDBP must issue notification that this has occurred. Thus the interface must include output conventions for such messages. Integrity control also verifies that new integrity constraints defined over a database are consistent with the existing data in the database.

Access control verifies that the issuer of an access request (retrieval or update) has been authorized by the owner of the data to make such a request. Performing this function requires that the LDBP component can identify the request issuer and his associated access rights. Access control also verifies that the issuer of a request to update access rights is authorized to make such changes. Whenever an access control constraint is violated, the LDBP should issue appropriate messages and record the violation.

Logical performance monitoring is the collection of statistics about the state and usage of a database in terms of its logical structure. The collection of these statistics can be invoked either implicitly by the receipt of an update that would affect them or explicitly by a request to scan the database or log files for this purpose. In the former case no special syntax is needed at the interface. In the latter case an invocation operation is needed.

Concurrency control coordinates the database interactions of users who are accessing a database at the same time. Concurrency control permits multiple users to access a database in a multi-programmed fashion while preserving the illusion that each user is executing alone on a dedicated system. The main technical difficulty in attaining this goal is to prevent database updates performed by one user from interfering with database retrievals and updates performed by another. To accomplish this function the LDBP component must know both the data to be accessed by each user and the times between which access to that data by other users might create problems. Syntax must be provided at the interface to express this information and to associate it with the appropriate invoking program.

Restructuring is required when changes are made to the structure description of a database. New data may need to be added to the stored representation of the database; existing data may need to be deleted or reorganized. These changes can be made in a single transaction where all other accesses against the database are restricted until the changes have been effected. Alternatively they may be made incrementally as distinct pieces of data are accessed in the course of normal operations.

Auditing is performed for purposes of security and reliability. Any operation on logical data objects can be recorded. These records can provide the basis for access trail analysis, statistics collection and, if need be,

recovery. This function, to be fulfilled satisfactorily, must be invoked automatically as required. Security and recovery based audit requirements must be specifiable by the DBA or other authorities. Security auditing can be specified by means of the trigger mechanisms of the Integrity Control Definition Language. Recovery auditing is also specified through the schema languages.

Optimization, in the most general case, uses information about access control and integrity constraints as well as existing access paths to select the best strategy for implementing an access request and monitoring performance. Depending upon the optimization strategies selected, an LDBP implementation may be able to exploit user directives to provide desired levels of performance. Such directives can be provided through the schema processors.

In summary, then, logical access, view mappings, integrity control, access control, performance monitoring requirements and concurrency control are all explicitly requested through the LDBP interface. Restructuring, auditing, and optimization, on the other hand, could be requested through logical updates on the schema of schemas.

In addition to performing the central task of mapping logical database access requests to calls on the physical database processor, the logical database processor also transforms and transmits data between its users and the physical database. Associated with it are four utilities that aid in this latter task: a logical loader, a logical dumper (unloader), a data translator, and a test data generator.

The Logical Loader

The logical loader reads source input data from machine-readable files and loads it into a database. It is intended to support rapid entry of large amounts of data into a database. Such a utility can be a simple fixed format processor or it can be generalized to handle input data in a variety of formats.

The Logical Dumper

The logical dumper unloads all or part of a database, as requested, onto an output or storage device. This utility dumps only logical objects. Dumpers at the physical database level and file level are proposed to support the dumping of lower level structures. The logical dumper and loader can be used to interchange data between standard DBMS's and programs external to the DBMS.

The Data Translator

A generalized data translator accepts descriptions of source data, target data, and a mapping from source to target. Following these descriptions it transforms the source data into the desired target format. It has several roles in a database management environment. The data translator can be used in logically or physically restructuring a database and translating a database from one DBMS to a database in a different DBMS. It may also be necessary in a heterogeneous distributed database management environment.

The Test Data Generator

Given information on database types and acceptable ranges of data values, a test data generator produces legal data objects. It is used in testing DBMS's and DBMS application programs.

3.2.4 The Physical Database Handling Processors

The central physical database handling processor is called the physical database processor. Its functionality is similar to that of the logical database processor. It differs primarily in that it processes physical rather than logical representations of the database. Associated with it are four utilities: a physical dumper, a physical loader (restorer), a physical structure checker, and a volume preconditioner.

The Physical Database Processor

The physical database processor performs six functions:

- physical access,
- physical performance monitoring,
- physical concurrency control,
- reorganization,
- physical auditing, and
- physical optimization.

Physical access is the mapping of physical data manipulation operations on the physical representation of a database to calls on the file access component.

Physical performance monitoring is the collection of statistics about the state and usage of a database in terms of its physical structure. The collection of these statistics can be invoked either implicitly by the receipt

of an update that would affect them or explicitly by a request to scan the database or log files for this purpose.

Physical concurrency control manages the execution of transactions at the physical structure level to guarantee the correctness of the database.

Reorganization is the modification of the physical database where its structure remains unchanged. It typically involves the reclaiming of fragmented space resulting from data deletions. It may also involve the reconstruction of directories and hash chains. Reorganization is usually initiated to improve performance.

Physical auditing is performed for purposes of reliability. Any operation on physical data objects can be recorded. These records can provide the basis for physical statistics collection and recovery.

Physical optimization is the buffering and other low level optimization of file access requests.

The Physical Dumper

The physical dumper utility is used to perform what is essentially a "core dump" of a database. It outputs the database in physical form. Thus, it makes directories and other physical access path data available for examination and/or correction by the DBA staff.

The Physical Loader

The physical loader reloads a (possibly modified) physical dump.

The Physical Structure Checker

The physical structure checker is a set of validation routines used to ensure that the physical database structures are valid. For example, in a CODASYL style DBMS they would be used to follow chains to verify that all owner pointers in a set point to the same owner.

The Volume Preconditioner

The volume preconditioner is a set of routines for formatting a volume (such as a disk pack) for use by a DBMS. This may include setting up tables and clearing space as necessary, depending on the physical requirements of the DBMS.

3.2.5 The File Access Processors

File access functionality is typically provided by operating systems. Standardization at this level could equally well be considered within the domain of operating system standards activities as within the domain of database standards activities. However, it is important to realize that database management imposes specific requirements on file systems. The performance and flexibility of the file system can have significant impact on the overall performance of any DBMS that it supports. For this reason, many DBMS vendors choose to provide their own file management routines rather than use those provided by the operating system of their host computer.

Only two file access processors have been included in the DBMS architecture: the file access processor itself and the file level dumper.

The File Access Processor

The file access processor executes file and device level data requests. These requests can include commands to create or destroy files and to read or delete records. The file system is also responsible for space and device management. It provides a variety of access structures (e.g. hashed placement, sequential placement).

The File Dumper

The file dumper performs database dumps at the physical file level. It makes no distinction between data objects and access path objects such as indices.

3.3 User Interfaces

The user interface processors are the components of the DBMS architecture that support the interactions between databases and their application programmers and end users. This class of components includes those for query language processing, host language interfacing and other more advanced user interfaces, such as natural language interfaces and tools such as editors to assist in the construction of user requests. These components and their position in the architecture are illustrated in Figure 3.3.

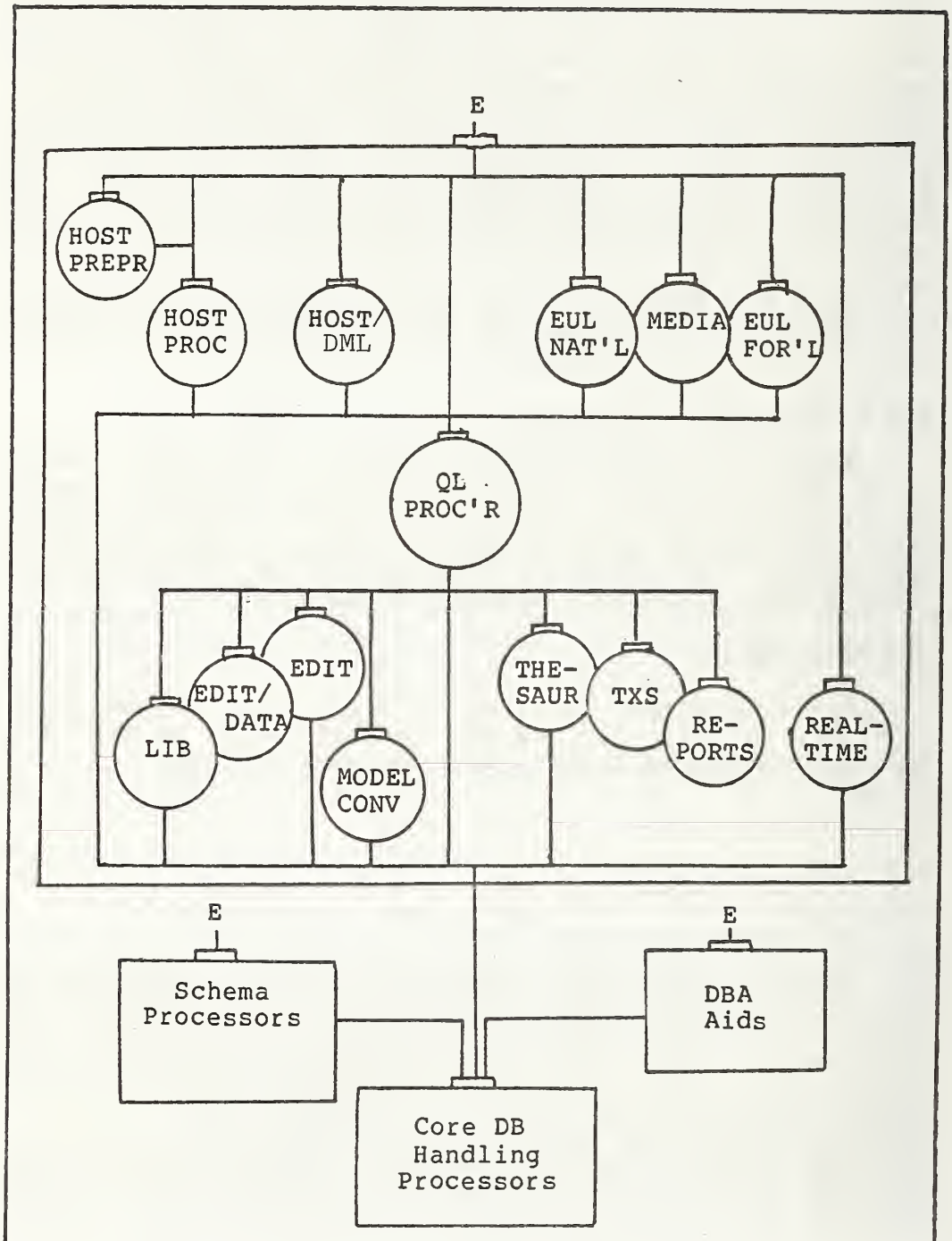


Figure 3.3 The User Interface Processors

The query language component is the most important of these components. This component is presented first, followed by a brief description of the remaining components in this class. These descriptions are organized into four separate sections: the host language interfaces, end user interfaces, special purpose interfaces, and utility processors. In the host language section, the host preprocessor (HOST PREPR), the host processor (HOST PROC) and the host language/data manipulation language processor (HOST/DML) are described. The natural language processor (EUL/NAT'L), formal language processor (EUL/FOR'L), and multi-media interface processor (MEDIA), are described in the section on end user language (EUL) interfaces. The thesaurus routines (THESAUR), text searching routines (TXS), and real-time data acquisition processors (REAL-TIME) are described in the section on special purpose interfaces. The final user interface section on utility processors describes the program library processor (LIB) and the various editors (EDIT, EDIT/DATA).

3.3.1 The Query Language Component

A query language (QL) is used to specify how database objects (items, entities and relationships) are retrieved, manipulated (inserted, deleted and modified) and how new objects are created. Appropriate retrieval and manipulation operations may be applied to and yield sets of database objects or single database objects. The "strawman" QL currently being developed will support both "set-at-a-time" and "record-at-a-time" processing.

The Level A QL components will include the functions needed for all data models: retrieval, update, and dynamic creation of structures. The functions will include:

a) retrieval of all instances of a record (entity) type,
b) retrieval of specified items from instances of a specified entity type,
c) boolean qualifications of selected entities,
d) qualification based on value set conditions,
e) retrieval involving set comparisons,
f) retrieval involving set operations,
g) retrieval involving qualification on multiple items in the same record,
h) retrieval involving the use of functions,
i) retrieval involving "cyclic" structures,
j) entity-at-a-time retrieval,
k) retrieval from more than one entity type,

l) simple value-based output ordering,
m) the storage, deletion and modification of entities,
n) assignment of retrieval results to a new entity type,
o) query composition,
p) transaction definition, and
q) integrity control invocation.

The QL would thus consist of entity selection, storage, deletion, modification, and integrity constraint invocation for the structures defined for Level A. A Level B QL would also include statements for connecting and disconnecting entities in Level B relationships.

One "strawman" Level B QL that is currently being investigated, will extend the Level A QL to allow direct operations on CODASYL sets, owner records, and member records. The extensions will include features for:

a) traversing a single CODASYL set,
b) traversing several CODASYL sets,
c) testing for empty sets,
d) hierarchic qualification,
e) traversing "link" records,
f) retrieving from sets with multiple member types,
g) retrieving from singular CODASYL sets,
h) retrieving from intra-record structures, and
i) altering membership in CODASYL sets.

3.3.2 Host Language Interfaces

The Host Language Interfaces are primarily intended for application programmers. They are the processors needed to provide programming facilities in which data manipulation operations are embedded in conventional general purpose programming languages (e.g., FORTRAN or COBOL). There are essentially three approaches for accomplishing this end: embedded subroutine calls, preprocessing and subroutine calls, and language extensions. The DBMS architecture was designed to accommodate all these approaches.

In the subroutine approach, data management routines are called from host language programs using conventional subroutine syntax. The second approach is a simple extension of the first where a preprocessor is provided to support more human engineered syntax. The subroutine and the preprocessor approaches are supported in the architecture by the two components called the host preprocessor and the host processor.

The Host Preprocessor

The preprocessor supports a syntax that can smoothly meld programming language syntax and database manipulation syntax. It converts statements in this syntax to pure host language statements and to calls on the logical database processor.

The Host Processor

The host processor is a (standard) general purpose programming language compiler.

In the language extension approach, modifications to accommodate integrated syntax are made directly to the host language compiler. These may take the form of developing a compiler for an entirely new language. This is the case for the research systems RIGEL [Rowe 1978] and PLAIN [Wasserman 1979]; or it may take the form of modification to an existing compiler as in the case of PASCAL R [Schmidt 1977].

The Host/DML Processor

The host language/data manipulation language processor is a compiler for a language that integrates the capabilities of a general purpose programming language and a data manipulation language.

In Figure 3.3, the host processor and the host language/data manipulation language processor are depicted as direct users of the interface to the core database handler. Alternately, they can interface to the query language processor to embed the syntax of the query language in a host programming language.

3.3.3 End User Interfaces

The end user interfaces support casual users and technical users. Casual users are those unwilling to learn or remember a technical query language or data manipulation language. Technical users are non-programmers who are accustomed to the discipline of using formal notations. Such users include engineers, scientists, technicians and accountants.

Four end user interfaces are included in the architecture by way of example. The fact that there are only four is not intended to imply that these exhaust all possibilities. It is within precisely these kinds of component subclasses that the DBMS architecture should

encourage innovation. The four components are: a query language processor (described in Section 3.3.1), a natural language processor, a formal language processor and a multi-media interface processor.

An End User Natural Language Processor

A natural language processor interacts with end users in a natural language (e.g., English). It converts user requests into DBMS requests via either the direct interface to the logical database processor or via an interface to one of the host language processors.

An End User Formal Language Processor

A formal language processor interacts with an end user in a very high-level but formatted language (e.g., through a forms notation).

The Multi-Media Interface Processor

The multi-media interface processor interacts with end users by means of a number of different devices. These may include graphical and audio output devices, and touch screen and joystick input devices.

3.3.4 Special Purpose Processors

The special purpose processors are included in the DBMS architecture to add application specific and "non-conventional" functionality to the DBMS user interfaces. For example, a text searching capability is a function usually associated with a document retrieval system and not a DBMS. However, there is a new and strong pressure created by a growing interest in automating offices to integrate into databases such diverse data types as documents, voice and pictures. Application specific components include those for business environments such as report writers and real-time environments.

Thesaurus Routines

Thesaurus routines accept values of "key" data items and produce synonyms that are legal database values or recommended retrieval values. They can also point users to related terms and otherwise allow users to expand their queries. The dictionaries needed to support this functionality can be managed by the DBMS just as it manages user and system (schema) databases.

Text Searching Routines

Text searching routines are used to implement conditions based on the contents of text data items (or collections of such items).

Report Writer

A report writer formats data retrieved from a database for output in the form of hard-copy reports. It supports options such as pagination, page headers, section divisions, columnar layouts, totals and subtotals.

Real-Time Data Acquisition Processors

A real-time data acquisition processor inputs data that is being generated in real-time to a DBMS. It provides buffering as needed.

3.3.5 End User Utilities

The end user utilities provide tools to users for creating and storing their queries and data manipulation requests.

Program Editor

A program editor provides standard editing capabilities that are tailored to the syntax of user interface languages.

Data Editor

A data editor is used to edit text items in database entities.

Program Library

A program library provides a centralized mechanism for storing and managing the software associated with user interfaces. This includes user applications programs and queries under development and in use. This software can be stored and managed by the DBMS just as it stores and manages user databases and other system databases.

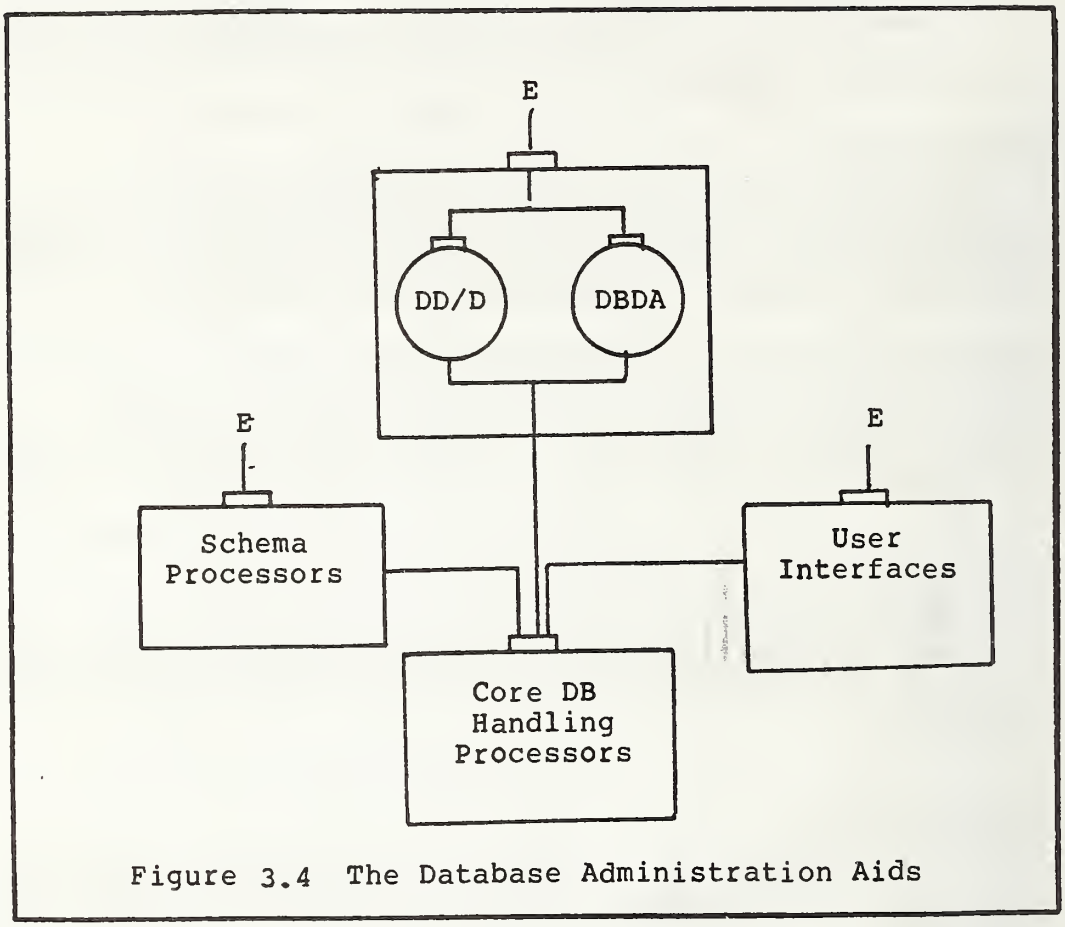


Figure 3.4 The Database Administration Aids

The database administration aids are essentially of two types: a documentation and management aid (a data dictionary/ directory component) and database design aids. The position of these aids in the DBMS architecture is depicted in Figure 3.4.

Data Dictionary/Directory

According to the British Computer Society (BCS), Data Dictionary Systems Working Party [BCS 1977], a data dictionary/directory (DD/D) is a tool for recording and processing information about the structure and usage of data. The information, that would appear in such a system, can range from the documentation of simple physical file structures to the design requirements, database structures and operation protocols of a large and sophisticated enterprise.

Database Design Aids

The database design aids are a set of routines for automatically producing logical and physical database designs. They can create candidate logical designs in order to study the effects of different designs on transaction and application programming. Similarly, other design aids can analyze the expected performance for physical database designs.

4. IMPACT OF THE ARCHITECTURE

The strawman architecture can have a significant impact on the future development of database management standards and systems. The benefits of adopting the architecture for the development of standards would be threefold.

- to provide an extensible reference model,
- to identify and prioritize interfaces, and
- to partition and coordinate standardization activities.

The first benefit of the component architecture is that it identifies a comprehensive set of DBMS and DBMS related functions and, through its family of components, supports multiple data models. The component architecture can thus serve as a reference model that represents a generic Database Management System and that can be used to describe and compare existing and future DBMS's. As a reference model, the architecture can be used to define more precisely the direct and indirect relationships among the different functions of a DBMS. As a reference model, the family of components can also be used to describe (and define) the similarities and differences between different data models.

The second benefit of the strawman architecture is that it identifies critical internal interfaces as candidates for standardization. The government and voluntary committees could standardize both the interface to the core database handler and the internal access to the schema of schemas. Specification of that interface supports the independent standardization and development of separate user and database administrator interfaces. The core database handler interface, together with standardized access to the schema of schemas, will permit user and third party vendor extensions to standardized DBMS's. Without such an interface, the original vendor must

develop all of the standard external interfaces for its own DBMS.

The third benefit of the strawman architecture is that it partitions the standardization process across components and data models. Such partitioning could have significant influence on voluntary standards committees and study groups. Each component of the architecture will have a uniform, minimal subset of syntax and functions that are common to the relational, hierarchical, network, and other data models. Each component can also have independent extensions to capture the significant differences between the models.

Under this scenario, the development of DBMS standards could be organized in a matrix-like fashion as shown in Figure 4.1. The Level A features would be defined and would serve as the portion of the reference model for standards common to all data models. A 'watchdog' body or technical committee could insure that standards to be developed for different models conform to the Level A specifications. In fact, for some components, all of the functionality for the different models may be standardized at Level A such that no further model dependent standardization is needed.

Standards could be developed that only address selected components. As a first step, one project or committee (say C1) could standardize the Level A definitions for the structure and integrity schemas in the schema of schemas and for logical database processor functions. Next a variety of standards projects or committees (i.e. C2, C3, C4 as shown in Figure 4.1) could proceed in parallel. For example, a technical committee could be chartered to provide the schema definition and data manipulation functions that are to be supported by the logical database processor. That interface would thus be independent of any specific host language.

In this way, the charter of existing and future committees could be specified in terms of the reference model. The existence of such a reference model permits a latitude in prioritizing and scheduling the development of particular standards that can better accommodate the desires of the different data model user communities. For example, one committee could have an initial charter as described above. That committee could standardize a network model COBOL interface without immediately trying to standardize a network model query language. Similarly, a relational query language could be standardized without standardizing the relational host language interface. Still other committees or groups could standardize a report writer facility for hierarchical systems. Each of

Model / Component	Level A	Level B			
		Relational	Hierarchical	Network	. . .
SCHEMA					
Structure	C1	C2	C3	C4	
Integrity	C1	C2	C3	C4	
Physical Structure					
.					
.					
.					
CORE DATABASE HANDLER					
Logical DBP	C1	C2	C3	C4	
Physical DBP					
File Access					
.					
.					
.					
USER INTERFACES					
Query Language		C2			
Report Writer			C3		
Host Language				C4	
.					
.					
.					
DBA AIDS					
Data Dictionary					
Design Tools					
.					
.					
.					

Figure 4.1 Standardization for DBMSs

these projects or committees would be standardizing level B extensions of standardized level A components.

In summary, the strawman architecture defines interfaces and families of components for multiple data models in order to provide a unified direction to the development of Database Management System standards. It is therefore imperative that Federal ADP departments, the computer industry and standardization groups provide a constructive analysis of the architecture.

5. REFERENCES

[BCS 1977]
British Computer Society. "The British Computer Society Data Dictionary Systems Working Party Report", Data Base, Vol. 9, No. 2, SIGMOD Record, Vol. 9, No. 4, December 1977.

[CW 1981]
Computerworld. March 16, 1981, p. 6.

[Date 1980]
Date, C.J. An Introduction to Database Systems, Addison-Wesley, 1980.

[Rowe 1978]
Rowe, L., and K. Shoens. "RIGEL: Preliminary Language Specification". Dept. Elec. Eng. and Comp. Sci., U.C. Berkeley. (December 1978).

[Schmidt 1977]
Schmidt, J. "Some High Level Language Constructs for Data of Type Relation". ACM Trans. on Data Base Sys., 2, 3, (September 1977), pp. 247-261.

[Wasserman 1979]
Wasserman, A.I. "The Data Management Facilities of PLAIN". ACM SIGMOD Conference, 1979, pp. 60-70.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. **BIBLIOGRAPHIC DATA SHEET** *(See instructions)*	1. PUBLICATION OR REPORT NO. SP 500-86	2. Performing Organ. Report No.	3. Publication Date January 1982

4. TITLE AND SUBTITLE

An Architecture for Database Management Standards

5. Prepared by:

Computer Corporation of America

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

Computer Corporation of America
575 Technology Square
Cambridge, MA 02139

7. Contract/Grant No.

NB79SBC0086

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

National Bureau of Standards
U.S. Department of Commerce
Washington, DC 20234

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 81-600174

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

This report describes the current status of an Institute for Computer Sciences and Technology project on architectures for Database Management Systems. An architectural framework for developing DBMS standards is presented. It addresses requirements of both the Federal data processing community and the DBMS vendor community. The architecture groups DBMS functions into both internal and external components and proposes for these components a family structure that supports the integration of DBMS standards for multiple data models.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

Database; database function; database management system; data model; schema; standards; system architecture; system components.

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D C 20402.

☐ Order From National Technical Information Service (NTIS), Springfield, VA 22161

14. NO. OF PRINTED PAGES

52

15. Price

$3.25

USCOMM-DC 6043-P80

☆U.S. GOVERNMENT PRINTING OFFICE: 1982-360-997/1922

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, D. C. 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. As a special service to subscribers each issue contains complete citations to all recent Bureau publications in both NBS and non-NBS media. Issued six times a year. Annual subscription: domestic $18; foreign $22.50. Single copy $4.25 domestic; $5.35 foreign.

NOTE: The Journal was formerly published in two sections. Section A "Physics and Chemistry" and Section B "Mathematical Sciences."

DIMENSIONS/NBS—This monthly magazine is published to inform scientists, engineers, business and industry leaders, teachers, students, and consumers of the latest advances in science and technology, with primary emphasis on work at NBS. The magazine highlights and reviews such issues as energy research, fire protection, building technology, metric conversion, pollution abatement, health and safety, and consumer product performance. In addition, it reports the results of Bureau programs in measurement standards and techniques, properties of matter and materials, engineering standards and services, instrumentation, and automatic data processing. Annual subscription: domestic $11; foreign $13.75.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396).

NOTE: The principal publication outlet for the foregoing data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Services, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Services, Springfield, VA 22161, in paper copy or microfiche form.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Washington, DC 20234